ReLU as a switch & associative memory

Think of a simple electrical switch controlling a wire:

- **Switch OFF** → no signal passes through (output = 0)
- **Switch ON** → the input signal passes through unchanged (output = input)

That's already very close to ReLU.

Mathematically, that switch is:

$$\text{output} = \begin{cases} x & \text{if switch is ON} \\ 0 & \text{if switch is OFF} \end{cases}$$

When ON, it's one-to-one: whatever comes in goes out unchanged.
When OFF, it kills the signal.

---

2. Now automate the switch

Instead of a human flipping it, suppose the switch flips automatically depending on the input:

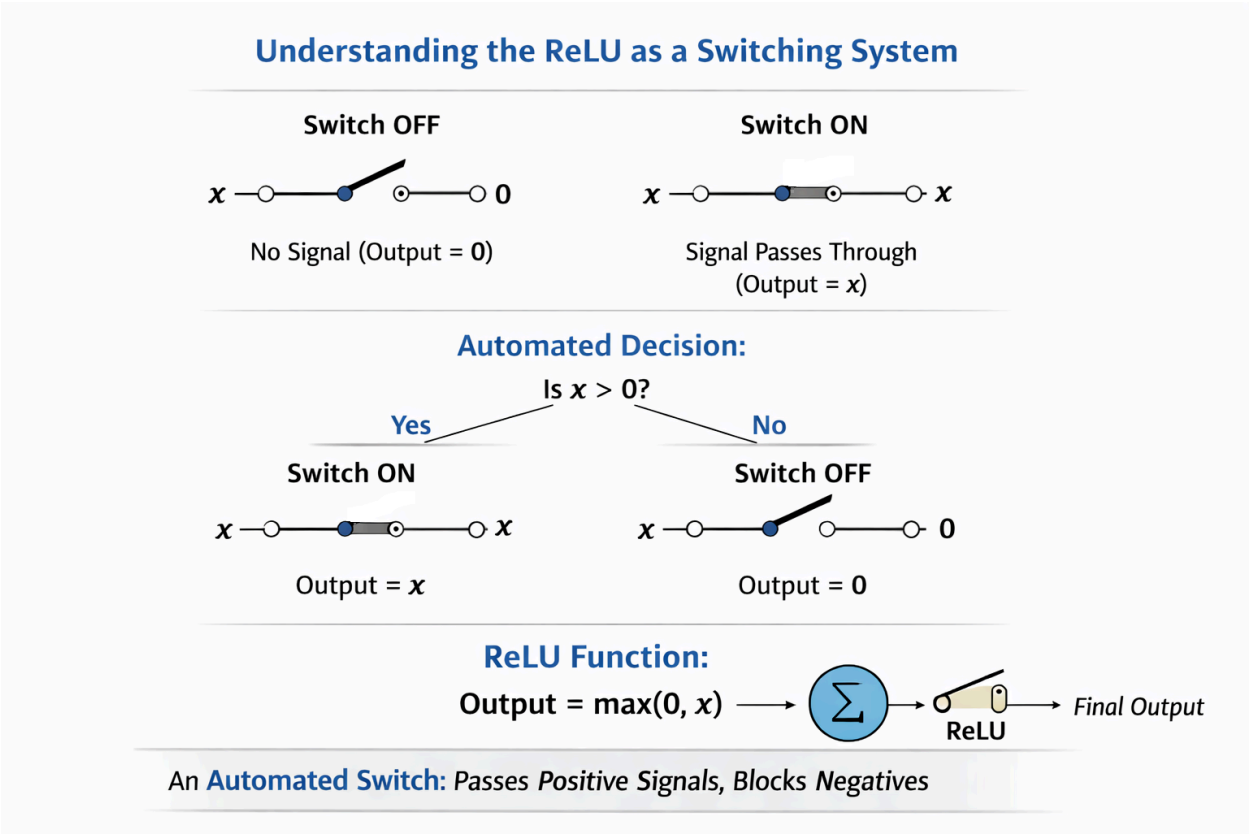
- If x>0 → switch turns ON
- If x≤0 → switch turns OFF

Now we have:

$$\text{output} = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

That *is* ReLU:

$$\text{ReLU}(x) = \max(0, x)$$



Neural Networks

Each neuron does:

1. Compute a weighted sum:
   $s = w \cdot x + b$
2. Pass that through ReLU:
   $y = \max(0, s)$

So each neuron is:

A weighted adder followed by an automatic switch.

Each neuron decides:

- “Am I ON?”
- “Or am I OFF?”

So the whole network is a huge collection of little automatic switches.

For a given input:

- Some neurons are ON (linear pass-through)
- Some neurons are OFF (zeroed out)

Once you fix which neurons are ON and OFF, the network becomes purely linear.

Why?

Because when ON, ReLU is identity.
When OFF, it's zero.

So for a fixed switch configuration:

> The network is just a big linear map.

A ReLU network is:

> A collection of linear functions, each selected by a pattern of switches.

Each ON/OFF pattern corresponds to a region of input space.

So the network is:

- Piecewise linear
- Switched linear systems
- A composition of gated weighted sums

You can say:

> A ReLU network is a dynamically reconfigurable linear circuit.
> The input decides which subcircuit is active.

Once the switches are fixed, everything collapses to a single linear map.

A single ReLU neuron does:

$$y = \text{ReLU}(w \cdot x + b)$$

Which is:

$$y = \begin{cases} w \cdot x + b & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Once you know whether it is ON or OFF:

- If OFF → output is 0.
- If ON → output is just a linear function of input.

So a single neuron is either:

- Zero, or
- A linear function.

Nothing nonlinear remains once the switch state is fixed.

Now consider a whole layer:

y=ReLU(Wx+b)

Let's define a diagonal matrix D that encodes which neurons are ON:

- Dii=1 if neuron i is ON
- Dii=0 if neuron i is OFF

Then the ReLU layer can be written as:

y=D(Wx+b)y

Now here's the key:

If D is fixed (i.e., the switching pattern is known), this is just matrix multiplication.

No nonlinearities left.

It becomes:

$y=DWx+Db$

That's linear in x.

Suppose you have two layers:

$y=D_2(W_2(D_1(W_1x+b_1))+b_2)$

If D_1 and D_2 are fixed (meaning we know which switches are ON), then everything is just matrix multiplication.

Expand it step by step:

$y=D_2W_2D_1W_1x+D_2W_2D_1b_1+D_2b_2$

Group the terms:

$y=Ax+cy$

Where:

$A=D_2W_2D_1W_1 \quad c=D_2W_2D_1b_1+D_2b_2$

That's just a single affine transformation.

For L layers, once all switches are known:

$y=D_LW_LD_{L-1}W_{L-1}\ldots D_1W_1x+\text{bias terms}$

All the D matrices are just constant diagonal matrices.

So the entire network reduces to:

$y=Ax+c$

A single weighted sum acting directly on the input.

Connection to Associative Memory

1. Start from the switch fact

A ReLU network is:

- Weighted sums
- Followed by automatic switches
- Repeated hierarchically

For any fixed input:

- Some neurons are ON
- Some are OFF
- The network collapses to a single affine map

So what varies across inputs is not the linear computation —
what varies is the **pattern of switches**.

That switch pattern is the key object.

---

2. View the switch pattern as a memory address

Each ReLU decision:

$w_i \cdot x + b_i > 0$?

is a binary test.

Across the whole network, you get a long binary vector:

$\sigma(x) \in \{0,1\}^N$

This binary pattern:

- Depends on the input
- Determines which linear map is active
- Selects which subcircuit runs

That is exactly what a memory address does.

So instead of saying:

> The network computes a nonlinear function

you can say:

> The network uses the input to select a stored linear transformation.

---

3. Why this is associative memory

Associative memory means:

- You don’t index memory by location.
- You index memory by similarity to content.

Here:

- The input itself determines the switch pattern.
- Similar inputs activate similar switch patterns.
- Similar switch patterns correspond to similar linear maps.

So the network does:

Input → switch pattern → selected linear map → output

The switch pattern is a **content-derived key**.

That's associative retrieval.

---

4. Why it is hierarchical

Because:

Layer 1 decides coarse features.
Layer 2 makes decisions conditioned on layer 1.
Layer 3 makes decisions conditioned on layers 1 and 2.

Each layer gates the next.

So the memory addressing is not flat.

It is progressively refined.

You can think of it as:

- First layer: broad partition of space
- Second layer: subpartition inside that region
- Third layer: finer subpartition

Each layer narrows the region of input space.
Each narrowing multiplies the gating matrices:

$A=D_L W_L \ldots D_1 W_1$

So the final linear map is not chosen in one step —
it is constructed hierarchically.

That is hierarchical memory addressing.

---

5. What is being stored?

Not examples.

Not prototypes.

What is stored is a collection of linear maps.

Each switch pattern corresponds to:

$y=A_{\sigma}x+c_{\sigma}$

So the network stores:

- A large set of linear transformations
- Organized by a hierarchy of gates
- Selected by input-dependent tests

That's a geometric memory.

---

6. Why this explains generalization

Within a region where the switch pattern is constant:

- The network behaves perfectly linearly
- Generalization is smooth
- The geometry is stable

When test data falls into a previously learned switch region:

- It activates the same subcircuit
- It retrieves the same linear transformation
- It generalizes correctly

So learning shapes the geometry of regions.
Inference selects among them.

That's associative recall of a linear operator.

---

7. Why this framing is stronger than metaphor

This is not analogy.
It is structural.

Because:

- ReLU = binary gating
- Binary gating defines regions
- Regions define linear maps
- Linear maps compose hierarchically

Therefore:

> A deep ReLU network is a hierarchical system that stores and retrieves linear operators using input-dependent gating decisions.

That is literally a hierarchical associative memory over geometric transformations.